Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 7

Тема: Проектирование структуры классов

Студент: Пивницкий Даниэль Сергеевич
Группа: 80-206
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Спроектировать простейший «графический» векторный редактор.

Требование к функционалу редактора:

- создание нового документа
- импорт документа из файла
- экспорт документа в файл
- создание графического примитива (согласно варианту задания)
- удаление графического примитива
- отображение документа на экране (печать перечня графических объектов и их характеристик в std::cout)
- реализовать операцию undo, отменяющую последнее сделанное действие. Должно действовать для операций добавления/удаления фигур.

Требования к реализации:

Создание графических примитивов необходимо вынести в отдельный класс – Factory.

Сделать упор на использовании полиморфизма при работе с фигурами;

Взаимодействие с пользователем (ввод команд) реализовать в функции main;

2. Описание программы

Основу программы составляет класс Document, реализующий методы добавления и удаления фигуры, вывода буфера на экран и в файл, отмены последнего действия. Также реализованы классы фигур с общим абстрактным родителем и обработка ввода.

3. Набор тестов

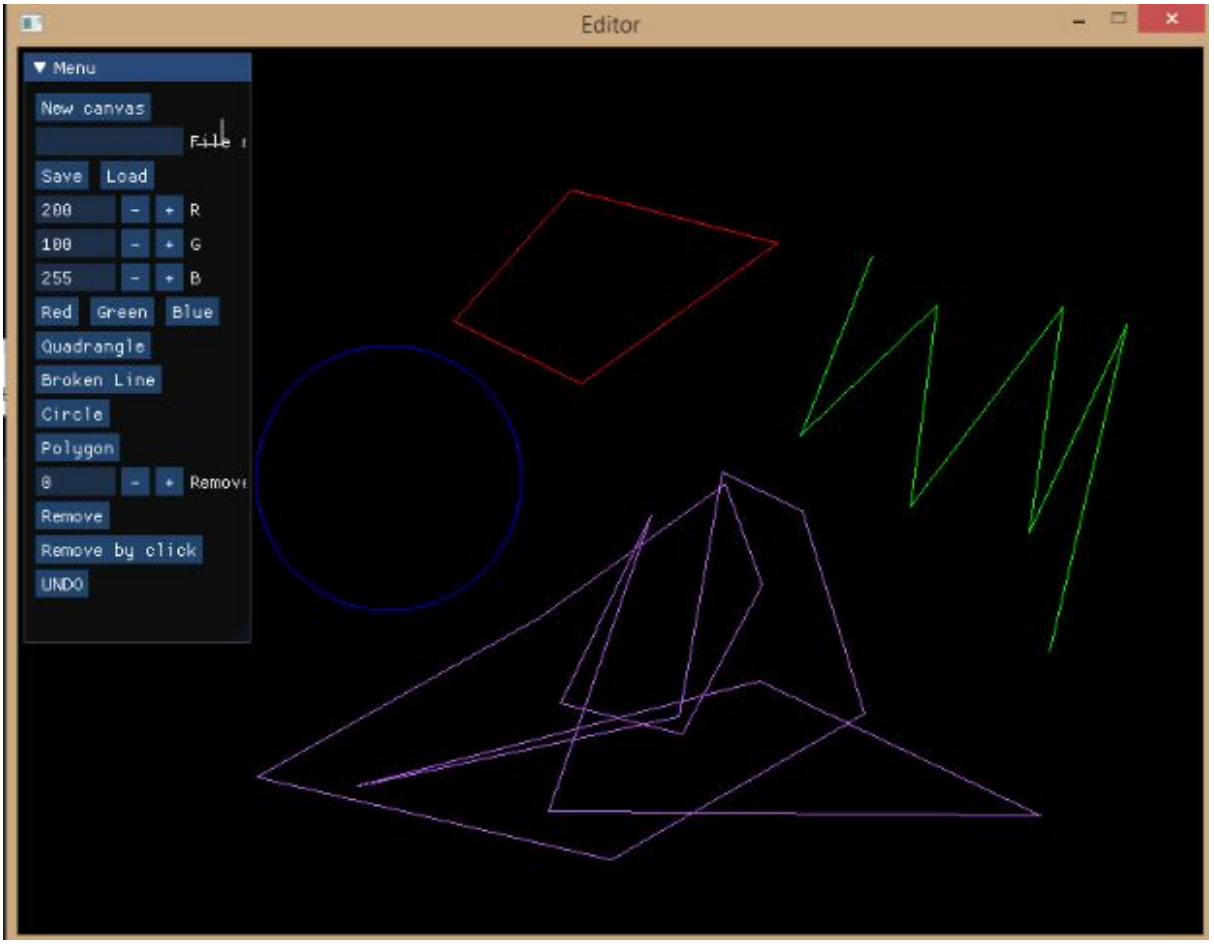
Тест представляет собой набор действий, проверяющий работоспособность каждого из методов класса Document в различных ситуациях.

test_1

Trapeze 1 1 2 2 3 3
Pentagon 4 4 5 5
Rhombus 4 4 5 5 6 6
Trapeze 7 7 7 7 7 7
Print
Undo

Print
Undo
Print
Remove 1
Print
Undo
Print
Undo
Print
Export temp.txt
Import temp.txt
Print

4. Результаты выполнения тестов



5. Листинг программы

```
//
//  main.cpp
//  lab7
//  Variant 15
//  M8o-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
```

```
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
/*
Спроектировать простейший «графический» векторный редактор.
Требования к реализации:
Создание графических примитивов необходимо вынести в отдельный класс – Factory.
Сделать упор на использовании полиморфизма при работе с фигурами;
Взаимодействие с пользователем (ввод команд) реализовать в функции main;
*/
#include <array>
#include <memory>
#include <vector>
#include <stack>

#include "sdl.h"
#include "imgui.h"

#include "triangle.h"
#include "hexagon.h"
#include "octagon.h"
#include "Document.h"

int main() {
	sdl::renderer renderer("Editor");
	bool quit = false;

	std::unique_ptr<builder> active_builder = nullptr;
	bool active_deleter = false;
	const int32_t file_name_length = 128;
	char file_name[file_name_length] = "";
	int32_t remove_id = 0;
	std::vector<int> color(3);

	Document currentDocument;

	while (!quit) {

		renderer.clear();

		sdl::event event;

		while (sdl::event::poll(event)) {
			sdl::quit_event quit_event;
			sdl::mouse_button_event mouse_button_event;
			if (event.extract(quit_event)) {
				quit = true;
				break;
			} else if (event.extract(mouse_button_event)) {
				if (active_builder && mouse_button_event.button() ==
```

```
sdl::mouse_button_event::left       &&       mouse_button_event.type()       ==
sdl::mouse_button_event::down) {
                                    std::unique_ptr<figure>         figure         =
active_builder->add_vertex(vertex{mouse_button_event.x(),
mouse_button_event.y()});
                                    if (figure) {
                                          figure -> setColor(color);


currentDocument.addFigure(std::move(figure));

                                          active_builder = nullptr;
                                    }
                              }
                              if (active_builder && mouse_button_event.button() ==
sdl::mouse_button_event::right      &&      mouse_button_event.type()      ==
sdl::mouse_button_event::down) {
                                    std::unique_ptr<figure>         figure         =
active_builder->add_vertex(vertex{-1, -1});
                                    if (figure) {
                                          figure -> setColor(color);


currentDocument.addFigure(std::move(figure));

                                          active_builder = nullptr;
                                    }
                              }

                        }
                  }

                  currentDocument.render(renderer);




                  ImGui::Begin("Menu");
                  if (ImGui::Button("New canvas")) {
                        currentDocument.clear();
                  }

                  ImGui::InputText("File name", file_name, file_name_length - 1);

                  if (ImGui::Button("Save")) {
                        std::ofstream os(file_name);

                        if (os) {
                              currentDocument.Save(os);
                        }
```

```
            }

            ImGui::SameLine();

            if (ImGui::Button("Load")) {
                  std::ifstream is(file_name);

                  if (is) {
                        currentDocument.Load(is);
                  }

            }



                  color[0] = 255;
                  color[1] = 255;
                  color[2] = 255;


            if (ImGui::Button("Triangle")) {
                  active_builder = std::make_unique<triangle_builder>();
            }
            if (ImGui::Button("Hexagon")) {
                  active_builder = std::make_unique<hexagon_builder>();
            }
            if (ImGui::Button("Octagon")) {
                  active_builder = std::make_unique<octagon_builder>();
            }

            ImGui::InputInt("Remove id", &remove_id);
            if (ImGui::Button("Remove")) {
                  if (remove_id >= 0 && remove_id <
(currentDocument.figures).size()) {

                        currentDocument.removeFigure(remove_id);

                  }
            }

            if (ImGui::Button("UNDO")) {

                  currentDocument.undo();
            }

            ImGui::End();

            renderer.present();
      }


}
```

```
//
//  octagon.cpp
//  lab7
//  Variant 15
//  M8o-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#include"octagon.h"

octagon::octagon(const std::array<vertex, 8>& vertices) : vertices_(vertices)
{}


void octagon::setColor(std::vector<int> color) {
    for (int i = 0; i < 3; i++) {
        color_.push_back(color[i]);
    }
}

void octagon::render(const sdl::renderer& renderer) const {
    renderer.set_color(color_[0], color_[1], color_[2]);
    for (int32_t i = 0; i < 8; ++i) {
        renderer.draw_line(vertices_[i].x, vertices_[i].y,
            vertices_[(i + 1) % 8].x, vertices_[(i + 1) % 8].y);
    }

}

void octagon::save(std::ostream& os) const {
    os << "octagon\n";
    for (int32_t i = 0; i < 8; ++i) {
        os << vertices_[i].x << ' ' << vertices_[i].y << '\n';
    }
    os << color_[0] << ' ' << color_[1] << ' ' << color_[2] << '\n';
}



std::unique_ptr<figure> octagon_builder::add_vertex(const vertex& v) {
    vertices_[n_] = v;
    n_ += 1;
    if (n_ != 8) {
        return nullptr;
    }
    return std::make_unique<octagon>(vertices_);
}
```

```
std::string octagon_builder::getType() {
    return "octagon";
}
//
//  hexagon.cpp
//  lab7
//  Variant 15
//  М8о-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#include"hexagon.h"

hexagon::hexagon(const std::array<vertex, 6>& vertices) : vertices_(vertices)
{}


void hexagon::setColor(std::vector<int> color) {
    for (int i = 0; i < 3; i++) {
        color_.push_back(color[i]);
    }
}

void hexagon::render(const sdl::renderer& renderer) const {
    renderer.set_color(color_[0], color_[1], color_[2]);
    for (int32_t i = 0; i < 6; ++i) {
        renderer.draw_line(vertices_[i].x, vertices_[i].y,
            vertices_[(i + 1) % 6].x, vertices_[(i + 1) % 6].y);
    }

}

void hexagon::save(std::ostream& os) const {
    os << "hexagon\n";
    for (int32_t i = 0; i < 6; ++i) {
        os << vertices_[i].x << ' ' << vertices_[i].y << '\n';
    }
    os << color_[0] << ' ' << color_[1] << ' ' << color_[2] << '\n';
}


std::unique_ptr<figure> hexagon_builder::add_vertex(const vertex& v) {
    vertices_[n_] = v;
    n_ += 1;
    if (n_ != 6) {
        return nullptr;
    }
    return std::make_unique<hexagon>(vertices_);
}
```

```
std::string hexagon_builder::getType() {
    return "hexagon";
}
//
//  triangle.cpp
//  lab7
//  Variant 15
//  М8о-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#include"triangle.h"

triangle::triangle(const std::array<vertex, 3>& vertices) :
vertices_(vertices) {}



void triangle::setColor(std::vector<int> color) {
    for (int i = 0; i < 3; i++) {
        color_.push_back(color[i]);
    }
}

void triangle::render(const sdl::renderer& renderer) const {
    renderer.set_color(color_[0], color_[1], color_[2]);
    for (int32_t i = 0; i < 3; ++i) {
        renderer.draw_line(vertices_[i].x, vertices_[i].y,
            vertices_[(i + 1) % 3].x, vertices_[(i + 1) % 3].y);
    }

}

void triangle::save(std::ostream& os) const {
    os << "triangle\n";
    for (int32_t i = 0; i < 3; ++i) {
        os << vertices_[i].x << ' ' << vertices_[i].y << '\n';
    }
    os << color_[0] << ' ' << color_[1] << ' ' << color_[2] << '\n';
}




std::unique_ptr<figure> triangle_builder::add_vertex(const vertex& v) {
    vertices_[n_] = v;
    n_ += 1;
    if (n_ != 3) {
        return nullptr;
```

```
    }
    return std::make_unique<triangle>(vertices_);
}

std::string triangle_builder::getType() {
    return "triangle";
}



//
//  sdl.cpp
//  lab7
//  Variant 15
//  M8o-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#include "sdl.h"

#include <SDL.h>

#include "imgui.h"
#include "imgui_sdl.h"
#include "imgui_impl_sdl.h"

namespace sdl {

sdl::sdl() {
  SDL_Init(SDL_INIT_VIDEO);
}

sdl::~sdl() {
  SDL_Quit();
}

renderer::renderer(const std::string& window_name):
        window_(SDL_CreateWindow(window_name.data(),  SDL_WINDOWPOS_CENTERED,
SDL_WINDOWPOS_CENTERED,
        800, 600, 0)),
    renderer_(SDL_CreateRenderer(window_, -1, SDL_RENDERER_SOFTWARE)) {
  ImGui::CreateContext();
  ImGui_ImplSDL2_Init(window_);
  ImGuiSDL::Initialize(renderer_, 800, 600);
  ImGui_ImplSDL2_NewFrame(window_);
  ImGui::NewFrame();
}

renderer::~renderer() {
  ImGuiSDL::Deinitialize();
```

```
  ImGui::DestroyContext();
  SDL_DestroyRenderer(renderer_);
  SDL_DestroyWindow(window_);
}

void renderer::set_color(uint8_t r, uint8_t g, uint8_t b) const {
  SDL_SetRenderDrawColor(renderer_, r, g, b, 255);
}

void renderer::clear() const {
  SDL_RenderClear(renderer_);
}

void renderer::draw_line(int32_t x1, int32_t y1, int32_t x2, int32_t y2)
const {
  SDL_RenderDrawLine(renderer_, x1, y1, x2, y2);
}

void renderer::draw_point(int32_t x, int32_t y) const {
  SDL_RenderDrawPoint(renderer_, x, y);
}

void renderer::present() const {
  ImGui::Render();
  ImGuiSDL::Render(ImGui::GetDrawData());
  SDL_RenderPresent(renderer_);
  ImGui_ImplSDL2_NewFrame(window_);
  ImGui::NewFrame();
}

quit_event::quit_event(const SDL_QuitEvent& e): event_(e) {}

mouse_button_event::mouse_button_event(const    SDL_MouseButtonEvent&    e):
event_(e) {}

uint32_t mouse_button_event::type() const {
  return event_.type;
}

uint8_t mouse_button_event::button() const {
  return event_.button;
}

int32_t mouse_button_event::x() const {
  return event_.x;
}

int32_t mouse_button_event::y() const {
  return event_.y;
}
```

```
bool event::extract(quit_event& event) const {
  if(event_.type == SDL_QUIT){
    event = event_.quit;
    return true;
  }
  return false;
}

bool event::extract(mouse_button_event& event) const {
  if(event_.type == SDL_MOUSEBUTTONDOWN || event_.type == SDL_MOUSEBUTTONUP){
    event = event_.button;
    return true;
  }
  return false;
}

bool event::poll(event& e) {
  bool result = SDL_PollEvent(&e.event_);
  if(result){
    ImGui_ImplSDL2_ProcessEvent(&e.event_);
  }
  return result;
}


//
//  loader.cpp
//  lab7
//  Variant 15
//  М8о-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#include"loader.h"

std::vector<std::unique_ptr<figure>> loader::load(std::ifstream& is) {
        std::string figure_name;
        std::vector<std::unique_ptr<figure>> figures;
        while (is >> figure_name) {
            vertex v;
            if (figure_name == std::string("curve_line")) {
                std::vector<vertex> vertices;
                int count_v;
                is >> count_v;
                for (int i = 0; i < count_v; ++i) {
                    is >> v.x >> v.y;
                    vertices.push_back(v);
                }
                struct color buffcolor {};
                is >> buffcolor.r >> buffcolor.g >> buffcolor.b;
```

```
}

else if (figure_name == std::string("quadrangle")) {
    std::array<vertex, 4> vertices;
    for (int i = 0; i < 4; ++i) {
        is >> v.x >> v.y;
        vertices[i] = v;
    }
    struct color buffcolor {};
    is >> buffcolor.r >> buffcolor.g >> buffcolor.b;

    figures.emplace_back(std::make_unique<quadrangle>(vertices));
    (*figures[figures.size() - 1]).set_color(buffcolor);
}
else if (figure_name == std::string("circle")) {
    std::vector<vertex> vertices(2);
    is >> vertices[0] >> vertices[1];
    struct color buffcolor {};
    is >> buffcolor.r >> buffcolor.g >> buffcolor.b;

    figures.emplace_back(std::make_unique<circle>(vertices));
    (*figures[figures.size() - 1]).set_color(buffcolor);
}

else if (figure_name == std::string("triangle")) {
    std::array<vertex, 3> vertices;
    for (int i = 0; i < 3; ++i) {
        is >> v.x >> v.y;
        vertices[i] = v;
    }
    struct color buffcolor {};
    is >> buffcolor.r >> buffcolor.g >> buffcolor.b;

    figures.emplace_back(std::make_unique<triangle>(vertices));
    (*figures[figures.size() - 1]).set_color(buffcolor);
}
else if (figure_name == std::string("polygon")) {
    std::vector<vertex> vertices;
    int count_v;
    is >> count_v;
    for (int i = 0; i < count_v; ++i) {
        is >> v.x >> v.y;
        vertices.push_back(v);
    }
    struct color buffcolor {};
    is >> buffcolor.r >> buffcolor.g >> buffcolor.b;

    figures.emplace_back(std::make_unique<curve_line>(vertices));
    (*figures[figures.size() - 1]).set_color(buffcolor);
}
else if (figure_name == std::string("hexagon")) {
```

```
                std::array<vertex, 6> vertices;
                for (int i = 0; i < 6; ++i) {
                    is >> v.x >> v.y;
                    vertices[i] = v;
                }
                struct color buffcolor {};
                is >> buffcolor.r >> buffcolor.g >> buffcolor.b;

                figures.emplace_back(std::make_unique<hexagon>(vertices));
                (*figures[figures.size() - 1]).set_color(buffcolor);
            }
        }
        return figures;
    }
    loader::~loader() = default;


//
//  document.cpp
//  lab7
//  Variant 15
//  M8o-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#include "Document.h"


void Document::addFigure(std::unique_ptr<figure> fig) {
      figures.emplace_back(std::move(fig));

      commandStack.push(std::make_unique<CommandAdd>(figures.size()    -    1,
this));

}
void Document::removeFigure(int id) {
      commandStack.push(std::make_unique<CommandRemove>(id,      figures[id],
this));
      figures.erase(figures.begin() + id);

}
void Document::undo() {
      if (commandStack.size()) {

            commandStack.top() -> undo();

            commandStack.pop();

      }
}
```

```
void Document::render(const sdl::renderer& renderer) {
	for (const std::shared_ptr<figure>& figure : figures) {
		figure -> render(renderer);
	}
}

void Document::Save(std::ofstream& os) {
	for (const std::shared_ptr<figure>& figure : figures) {
		figure -> save(os);
	}
}

void Document::Load(std::ifstream& is) {

	figures.clear();

	while ( ! commandStack.empty() )
	{
	commandStack.pop();
	}

	std::string type;

	while(std::getline(is, type)) {
		if (type == "triangle") {
			std::array<vertex, 3> vrt;
			for (int i = 0; i < 3; i++) {
				is >> vrt[i];
			}
			std::vector<int> colorTmp(3);
			for (int i = 0; i < 3; i++) {
				is >> colorTmp[i];
			}
			std::unique_ptr<figure>		trap		=
std::make_unique<triangle>(vrt);
			trap->setColor(colorTmp);
			figures.emplace_back(std::move(trap));
		}
		else if (type == "hexagon") {
			std::array<vertex, 6> vrt;
			for (int i = 0; i < 6; i++) {
				is >> vrt[i];
			}
			std::vector<int> colorTmp(3);
			for (int i = 0; i < 3; i++) {
				is >> colorTmp[i];
			}
			std::unique_ptr<figure>		trap		=
std::make_unique<hexagon>(vrt);
```

```
                trap->setColor(colorTmp);
                figures.emplace_back(std::move(trap));
            }
            else if (type == "octagon") {
                std::array<vertex, 8> vrt;
                for (int i = 0; i < 8; i++) {
                    is >> vrt[i];
                }
                std::vector<int> colorTmp(3);
                for (int i = 0; i < 3; i++) {
                    is >> colorTmp[i];
                }
                std::unique_ptr<figure>              trap              =
std::make_unique<octagon>(vrt);
                trap->setColor(colorTmp);
                figures.emplace_back(std::move(trap));
            }

        }

}

void Document::clear() {

        while ( ! commandStack.empty() )
        {
        commandStack.pop();
        }

        figures.clear();

}
```

6. Выводы

Я познакомился с написанием и сборкой многофайловых проектов, а также подключением сторонних библиотек, углубил свои знания в использовании полиморфизма.

7. Список литературы

1.Руководство по написанию кода на C++ [Электронный ресурс].
URL:https://metanit.com/cpp/tutorial/
Дата обращения: 10.09.2019

2.Документация по C++ [Электронный ресурс]. URL:
https://docs.microsoft.com/ru-ru/cpp
Дата обращения 12.09.2019